Tyabandha Journal of Arts and Science

ISSN 1744–9669

Volume 4, Number 4

October 2007

Edited by Vaen Sryayudhya

Hadrian's Wall ... by Kit Tyabandha 315

C Programming on Unix, Part III ... by Kit Tyabandha 405

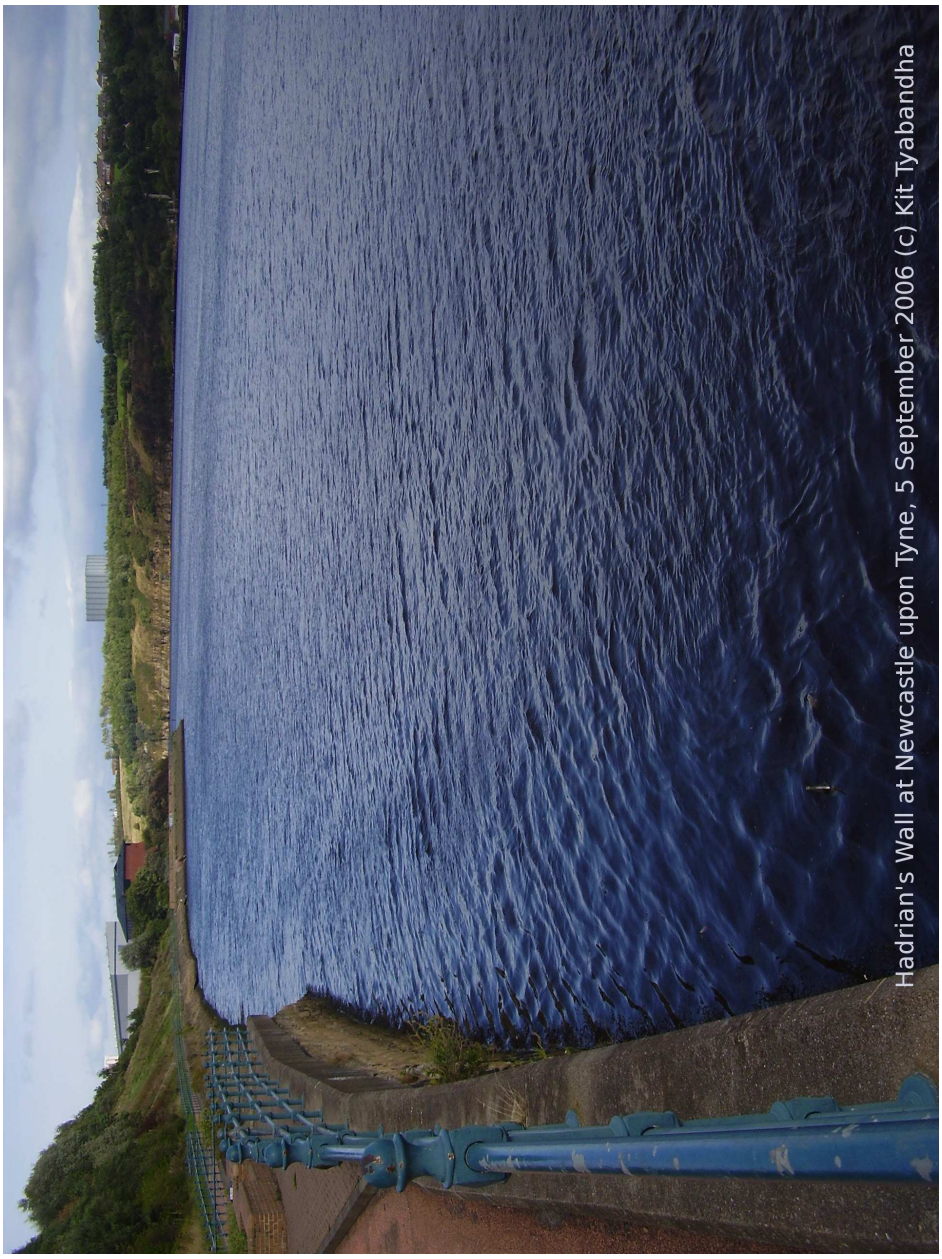
England
3rd February, 2008

Distribution
Internet-searchable world-wide

Hadrian's Wall

Kit Tyabandha, Ph.D.

Segedunum, Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha


Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

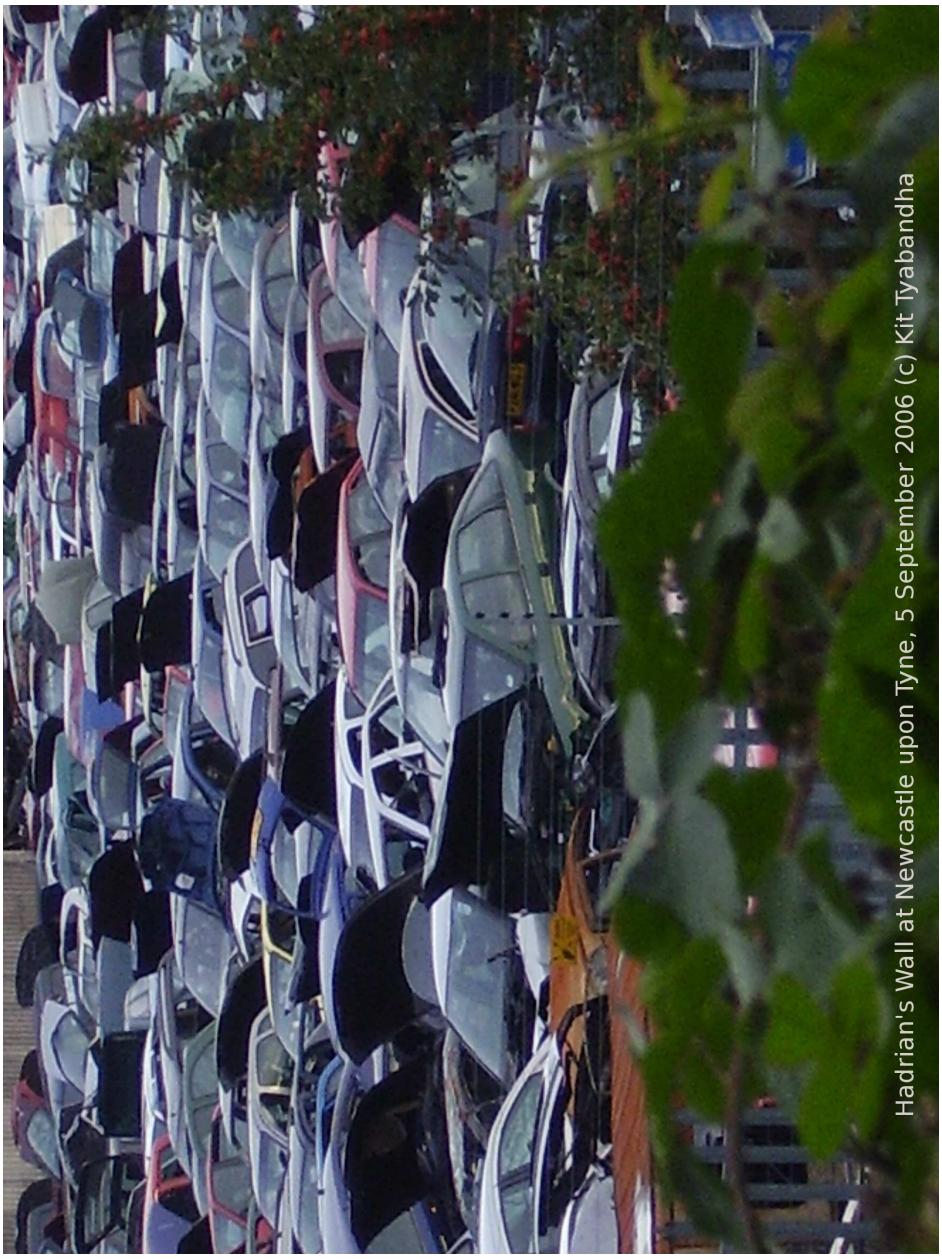
BALTIC FLOUR MILLS
Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Ki

Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

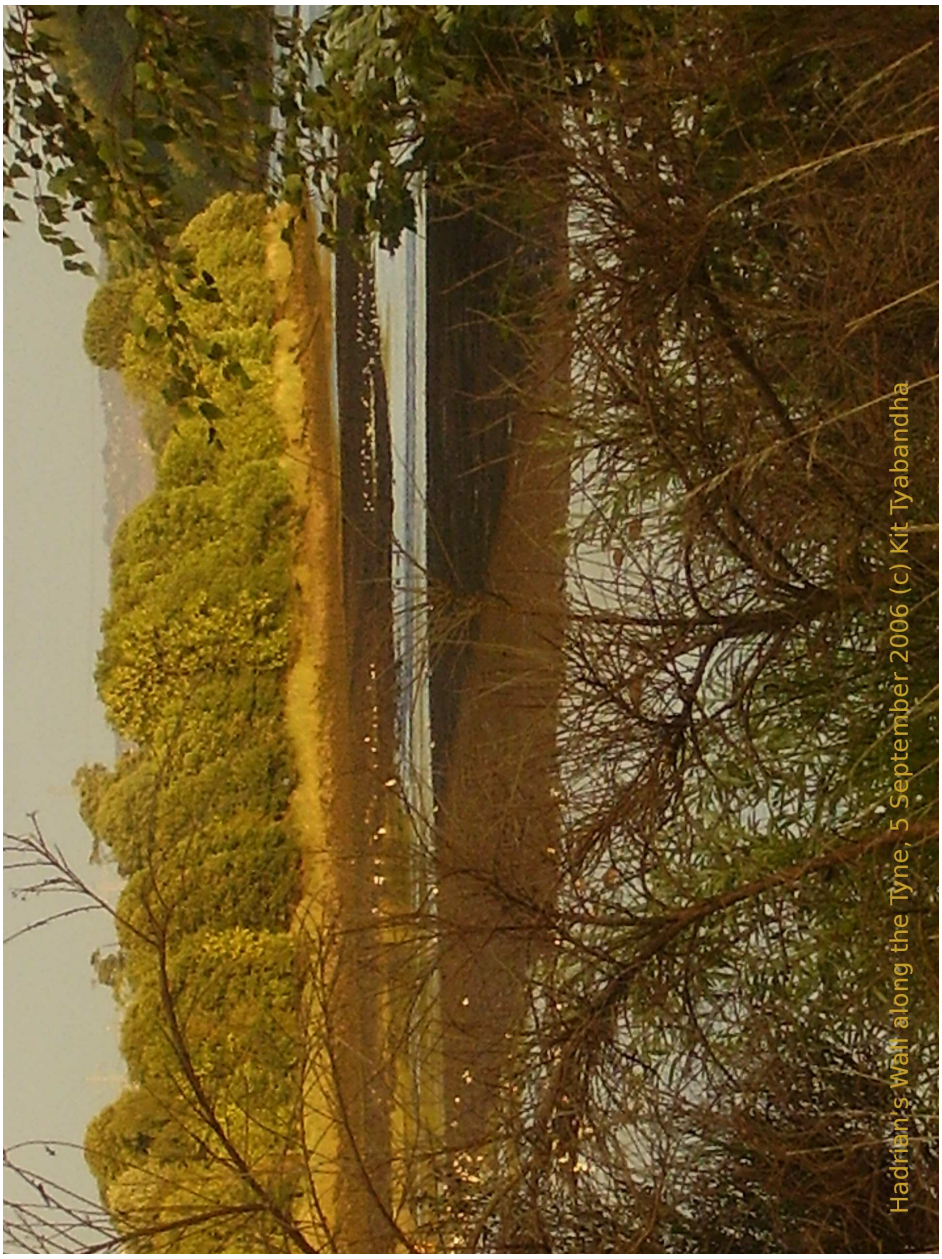
Hadrian's Wall at Newcastle, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle, 5 September 2006 (c) Kit Tyabandha


Hadrian's Wall at Newcastle upon Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Newcastle, 5 September 2006 (c) Kit Tyabandha

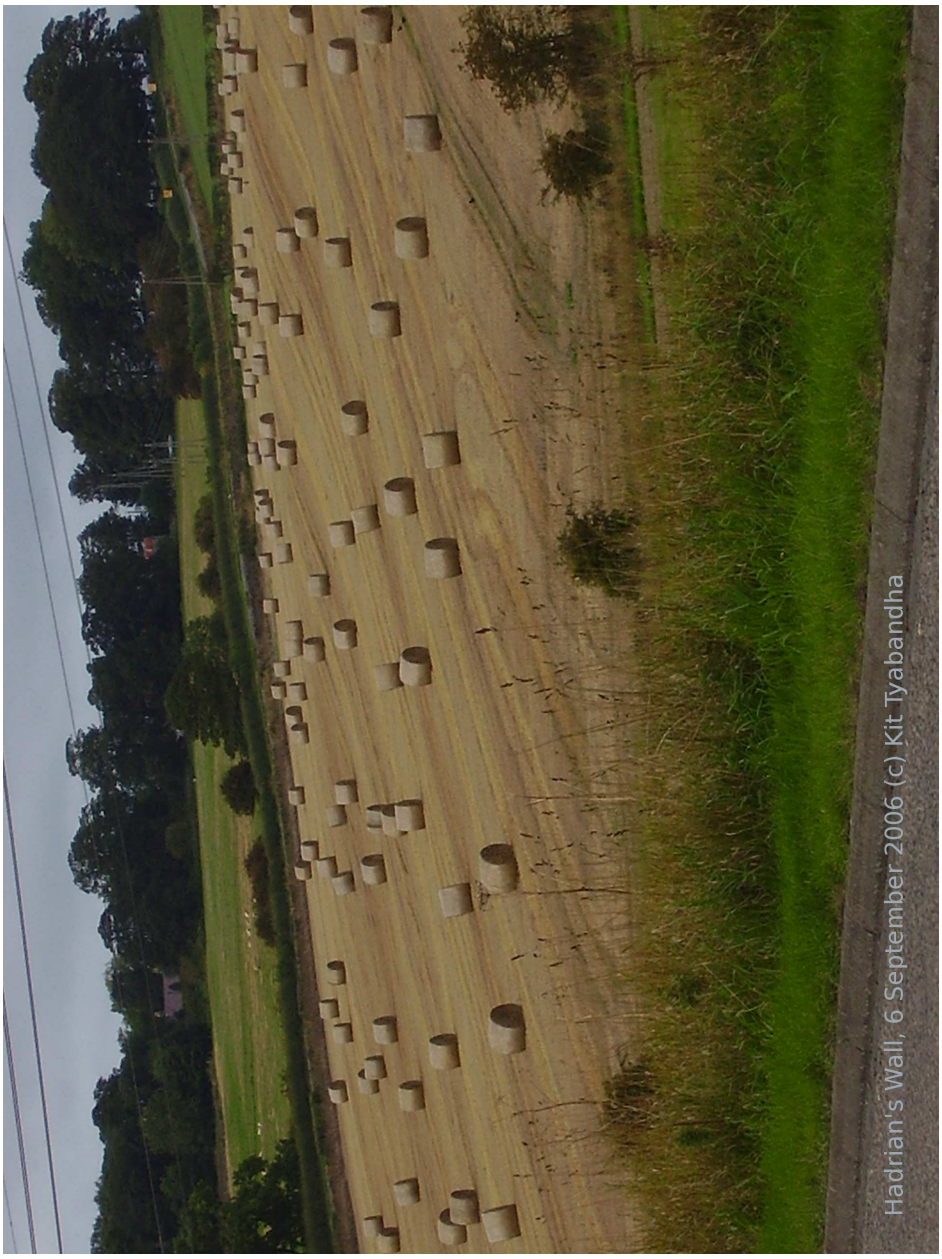
Hadrian's Wall along the Tyne, 5 September 2006 (c) Kit Tyabandha


Hadrian's Wall along the Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall on the Tyne, 5 September 2006 (c) Kit Tyabandha

Hadrian's Wall at Houghton North Farm, Heddon on the Wall
6 September 2006 (c) Kit Tyabandha

Houghton North Farm, Heddon on the Wall
Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha


Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

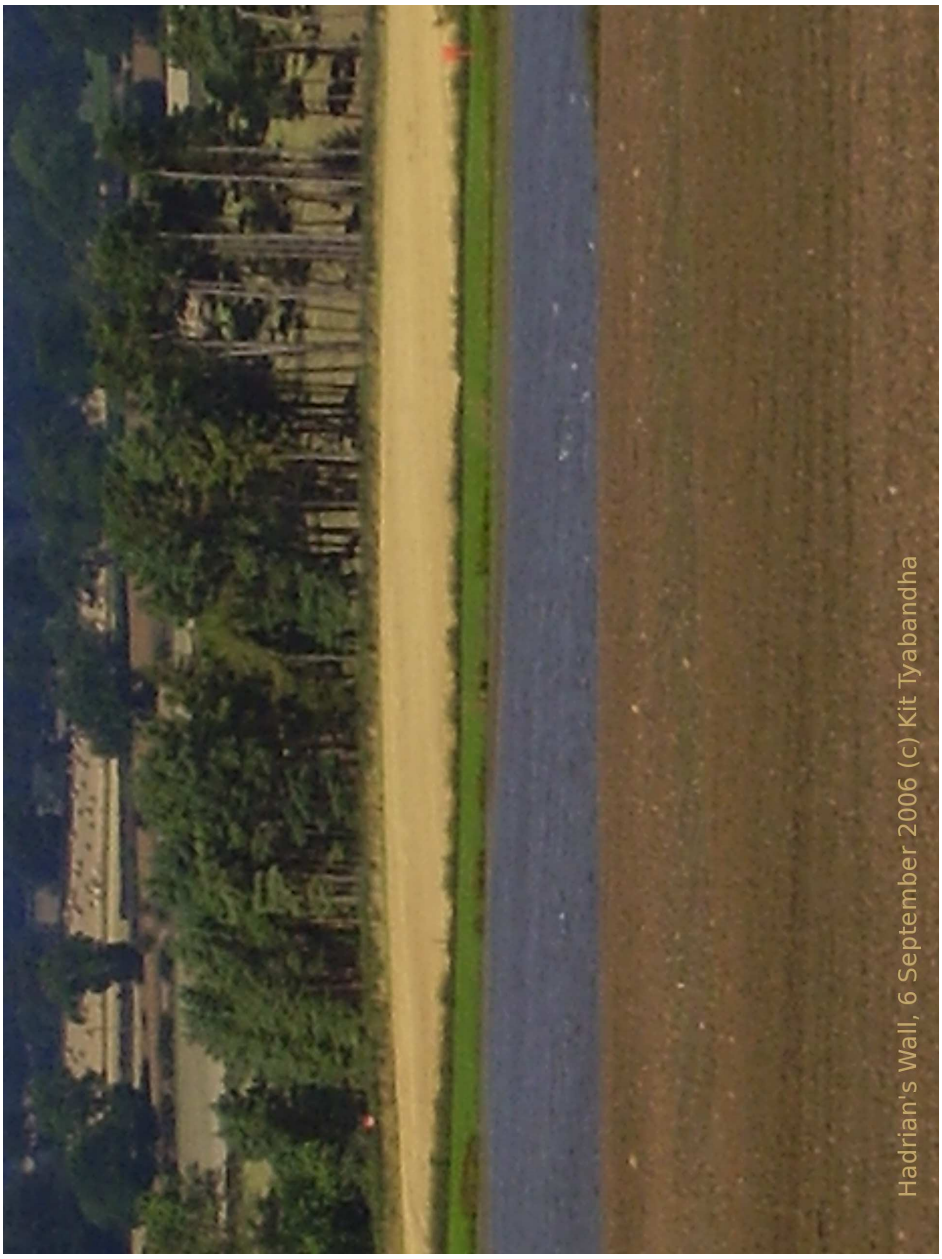
Hadrian's Wall, 6 September 2006, © Kit Tyabandha

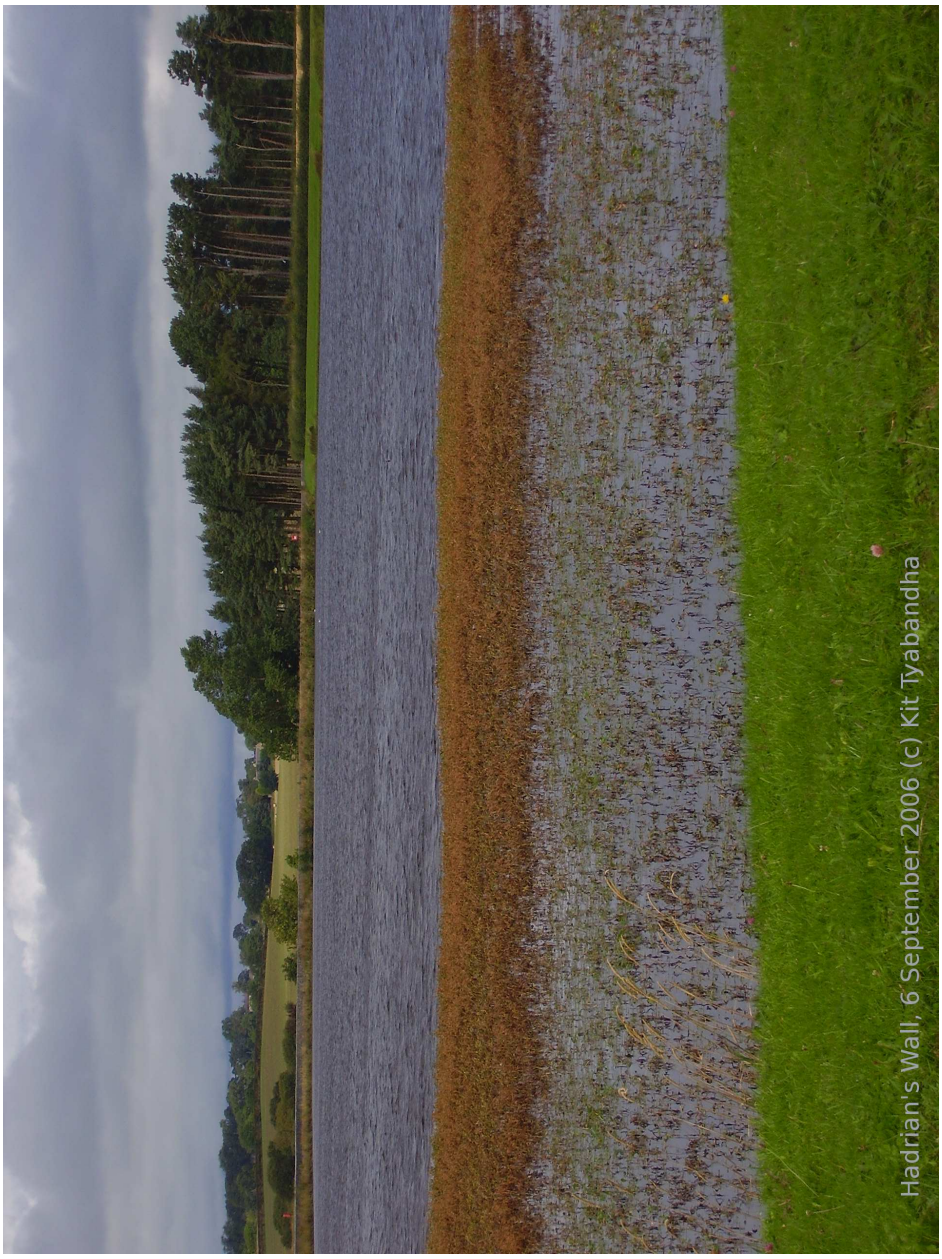
Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006, © Kirk Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall - Section
(c) Kit Tyabandha


Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

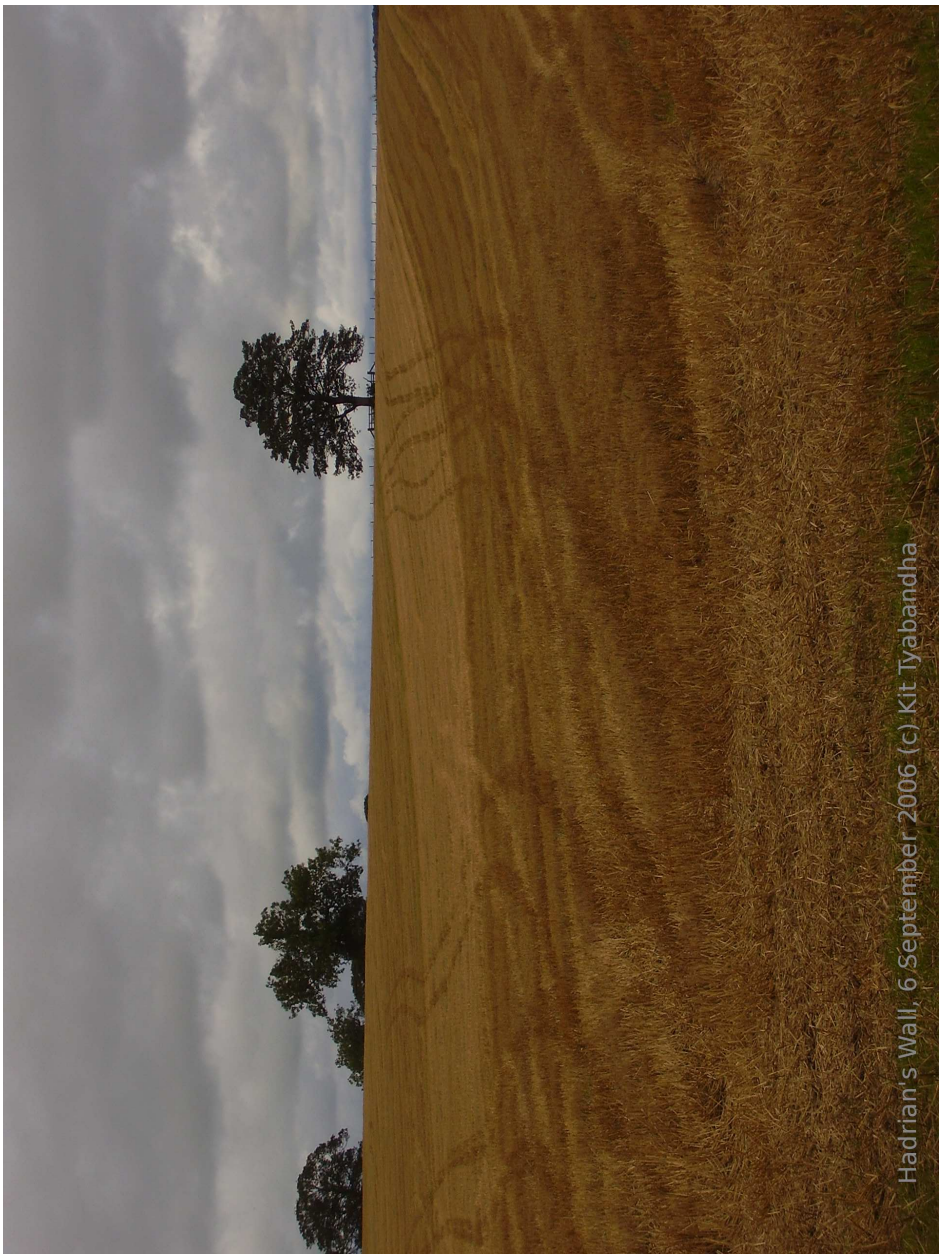

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

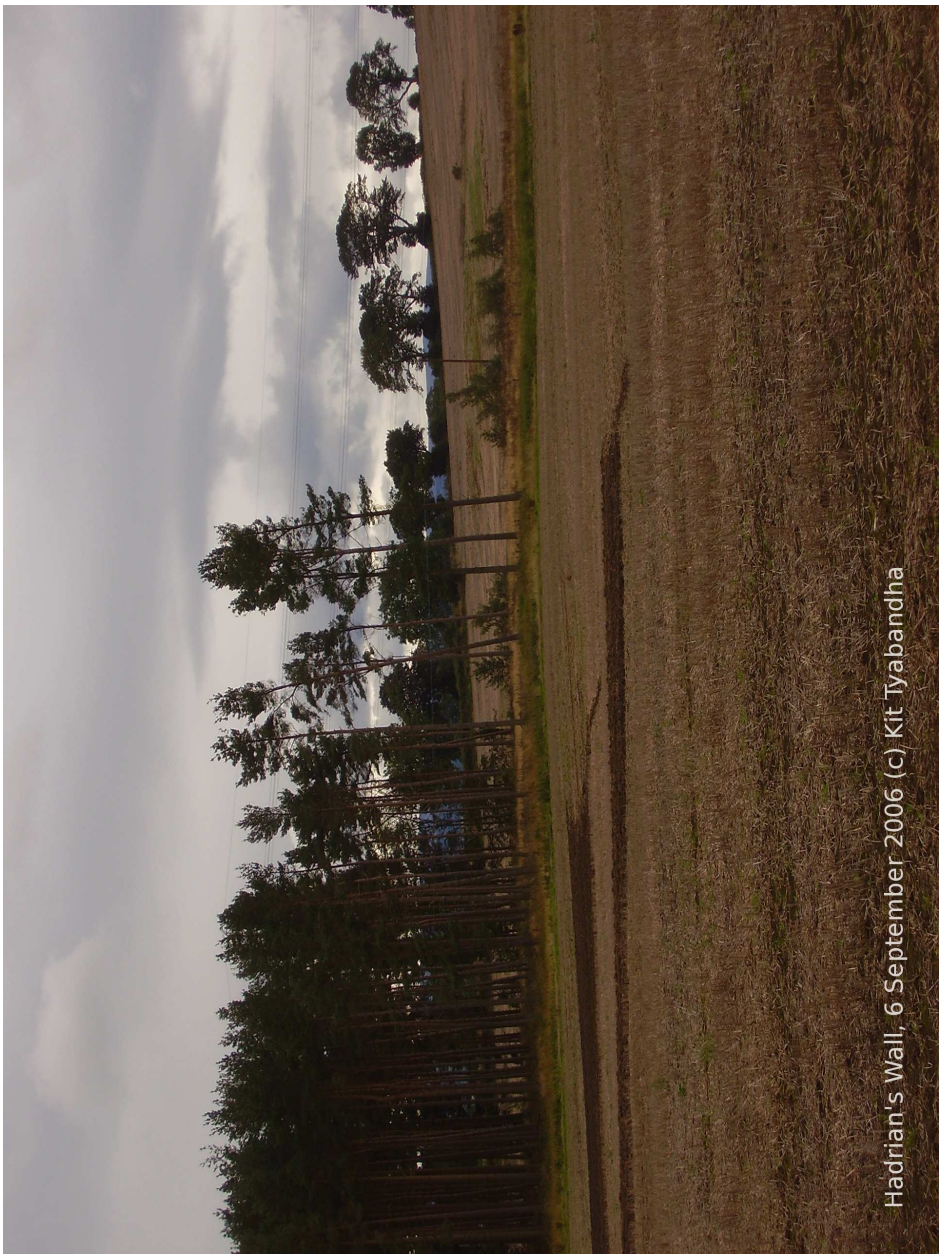
Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Rajdhani Vkd, 6 September 2006, Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (C) Kit Tyabandha

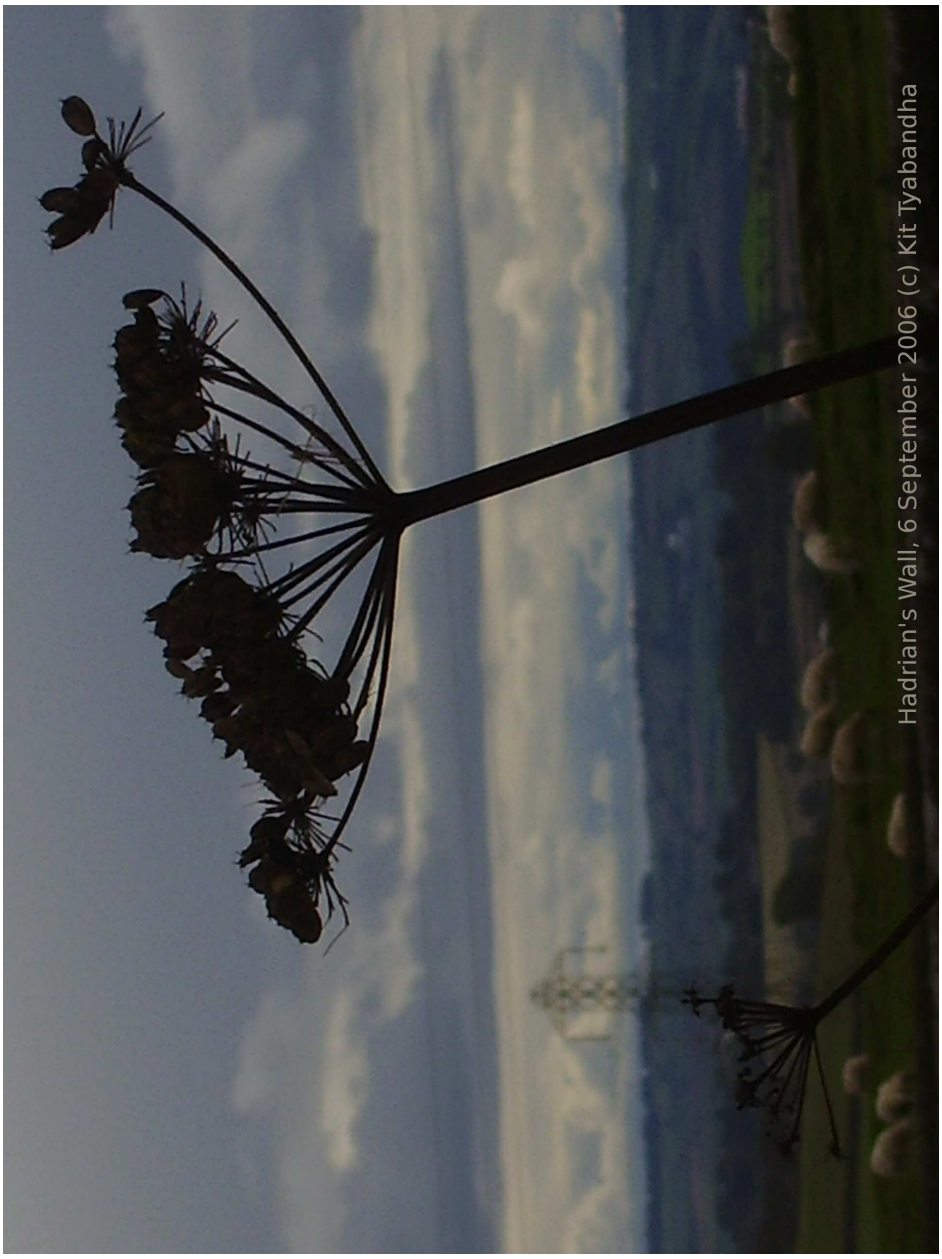

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha


Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha


Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 September 2006 (c) Kit Tyabandha

Hadrian's Wall, 6 Sept 2006 (c) Kit Tyabandha

C Programming on Unix Part III

Kit Tyabandha, Ph.D.

Lecture

16^{th} January 2006

Strịnondȧḅuriị

Sequential and non-sequential modes of output

There are two modes of screen display, one sequential the other non-sequential. We have been by now fairly familiar with the former. Programme 1 tests the latter.

Programme 1 *Programme to test non-sequential output*

```
1 /* programme to test non-sequential mode of output,
2    Kit Tyabandha, 14 Dec 2006 */
3 #include<curses.h>
4 #include<stdio.h>
5 int main(){
6   initscr();
7   move(20,20);
8   addstr("test");
9   getch();
10  endwin();
11  return 0;
12 }
```

We next look at how assembly codes are generated from each compilation. Programme 2 gives a simple code. We compile this programme with

```
gcc -c -g -Wall tst.c
gcc -g tst.o -o tst -lcurses -ldl -lm
```

our programme being *tst.c*. Listing 3 shows the Assembly code obtained from the compilation we did above.

Programme 2 *A simple programme*

```
/* looking at the assembly codes generated from our compilation,
   Kit Tyabandha, 18 Dec 2006 */
#include<stdio.h>
#include<stdlib.h>
int main(){
  int a, b, c;
  a =1;
  b =2;
  c =a+b;
  printf("\n c = %d\n\n",c);
  exit(0);
}
```

Listing 3 *Assembly code from a compilation*

```
.file "tst.c"
.section .rodata
.LC0:
.string "\n c = %d\n\n"
.text
.globl main
.type main, @function
main:
pushl %ebp
movl %esp, %ebp
subl $24, %esp
andl $-16, %esp
movl $0, %eax
subl %eax, %esp
movl $1, -4(%ebp)
movl $2, -8(%ebp)
movl -8(%ebp), %eax
addl -4(%ebp), %eax
movl %eax, -12(%ebp)
movl -12(%ebp), %eax
movl %eax, 4(%esp)
movl $.LC0, (%esp)
call printf
movl $0, (%esp)
call exit
.size main, .-main
.section .note.GNU-stack,"",@progbits
.ident "GCC: (GNU) 3.3.5 (Debian 1:3.3.5-3)"
```

Lecture

23^{rd} January 2006

Stṛinondȧḅuṛi

Header and macros

Header files contain a stock of library functions for common use. They are named with a `.h` extension. We declare the use of these library functions by including some of these files through the command `#include`. In particular, say if our header file is `header.h`, then `#include<header.h>` looks for the file in the place where it is expected, for example `/usr/include/` whereas `#include "header.h"` looks first for the file in the present working directory. Should that fail it would look next in `/usr/include`.

Example 1 and Programme 4 shows how a header file may be used. The `#ifndef` prevents against multiple inclusion of the same header as multiple declaration may confuse the compiler.

Example 1. (Header file example)

```
1 /* header1.h, a header file, Kit Tyabandha, 23 Jan 07 */
2 #ifndef header1_h
3 #define header1_h
4 #define max(a, b) ((a)>(b)?(a):(b))
5 #endif
```

Programme 4 *Inclusion of the header file in Example 1*

```
/* macros, Kit Tyabandha, 6 Feb 07 */
#include<stdio.h>
#include<stdlib.h>
#include "header1.h"
int
main(){
  int x=4, y=6;
  printf("\n x = %d, y = %d, max(x,y) = %d\n\n",\
    x, y, max(x,y));
  exit(0);
}
```

Header `header1.h` also shows one way to implement a macro. These are no sub-routines but expand according to their definition every time they are used in a programme. Following is the compilation and running of Programme 4.

```
kit@nebula:~/prog/c$ make tst
gcc -c -g -Wall tst.c
gcc -g tst.o -o tst -lcurses -ldl -lm
kit@nebula:~/prog/c$ tst
 x = 4, y = 6, max(x,y) = 6
```

Stringising operator

The stringising operator # encloses the argument of `#define` within double quotes. Example 2 gives a header file, `header2.h`, that uses #-operators to turn variable names into strings. Programme 5 uses the macro using this from the header file.

Example 2. (Header file using stringising operators)

```
1 /* header2.h, Kit Tyabandha, 6 Feb 07 */
2 #ifndef header2_h
3 #define header2_h
4 #define max(a, b)\
5    printf("\n "#a" = %.1f\n "#b" = %.1f\n max("#a", "#b") =\
6      %.1f\n\n", (a), (b), ((a)<(b)?(b):(a)))
7 #endif
```

Programme 5 *Use of the header file in Example 2*

```
1 /* stringising operator #, Kit Tyabandha, 6 Feb 07 */
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include "header2.h"
5 int
6 main(){
7   float x=8, y=6;
8   max(x,y);
9   exit(0);
10 }
```

When compiled and run the results were as shown in the following.

```
kit@nebula:~/prog/c$ make tst
gcc -c -g -Wall tst.c
gcc -g tst.o -o tst -lcurses -ldl -lm
kit@nebula:~/prog/c$ tst
 x = 8.0
 y = 6.0
 max(x, y) = 8.0
```

Lecture

30^{th} January 2006

Strịnondȧḅuṛị

Forking

Unix is a multi-programming system. The kernel brings the system up at booting and then it stops leaving its children, namely tasks and drivers, to run. All user processes are children of `init`.

The only way to create a new process in Unix is by using the `fork` command. This command is declared in the *unistd* header.

Example 3 demonstrates how `fork` works. Here the `if–else` conditional statement has two possibilities both of which run in parallel to each other. The *stdlib* header is there because `system` needs it, the *stdio* because `printf` does.

Example 3. (Example of forking and child processes)

```
1  /* two children running in parallel, Kit Tyabandha,
2    30 Jan 07 */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <unistd.h>
6  int
7  main(){
8    printf("\n Parent\n\n");
9    if(fork()!=0){
10     printf(" Child 1\n\n");
11     system("date");
12   } else {
13     printf(" Child 2\n\n");
14     system("whoami");
15   }
16   return 0;
17 }
```

Bibliography

Andrew S Tanenbaum and Albert S Woodhall. *Operating systems design and implementation.* 3rd ed., Prentice-Hall, 2006(1987)

Lecture

6^{th} February 2006

Strịnondȧḅuriṛ

Synonym

As a language, C also has synonyms. For example, in C array and pointer are interchangeable. Programme 6 shows four different ways to access an element of a matrix, through different words which mean the same thing. Notice that indices run beginning from zero not one.

Programme 6 *Array and pointer representations of a matrix*

```
1  /* synonyms, Kit Tyabandha, 6 Feb 07 */
2  #include <stdio.h>
3  int
4  main(){
5    int a[5][6]={
6      {11, 12, 13, 14, 15, 16},
7      {21, 22, 23, 24, 25, 26},
8      {31, 32, 33, 34, 35, 36},
9      {41, 42, 43, 44, 45, 46},
10     {51, 52, 53, 54, 55, 56}}, i=2, j=3;
11   printf("\n");
12   printf("a[i][j] =\n\t\
13     %d, %d, %d, %d, %d, %d\n\t\
14     %d, %d, %d, %d, %d, %d\n\t\
15     %d, %d, %d, %d, %d, %d\n\t\
16     %d, %d, %d, %d, %d, %d\n\t\
17     %d, %d, %d, %d, %d, %d\n\n",\
18     a[0][0], a[0][1], a[0][2], a[0][3], a[0][4], a[0][5],\
19     a[1][0], a[1][1], a[1][2], a[1][3], a[1][4], a[1][5],\
20     a[2][0], a[2][1], a[2][2], a[2][3], a[2][4], a[2][5],\
21     a[3][0], a[3][1], a[3][2], a[3][3], a[3][4], a[3][5],\
22     a[4][0], a[4][1], a[4][2], a[4][3], a[4][4], a[4][5]);
23   printf("a[%d][%d] = %d\n\n", i, j, a[i][j]);
24   printf("*(a[%d]+%d) = %d\n\n", i, j, *(a[i]+j));
25   printf("*(*(a+%d)+%d) = %d\n\n", i, j, *(*(a+i)+j));
26   printf("*(*a+%d*n+%d) = %d\n\n", i, j, *(*a+i*6+j));
27   return 0;
28 }
```

We compile and run this programme as the following.

```
kit@nebula:~/prog/c$ make tst
gcc -c -g -Wall tst.c
gcc -g tst.o -o tst -lcurses -ldl -lm
kit@nebula:~/prog/c$ tst
a[i][j] =
            11, 12, 13, 14, 15, 16
            21, 22, 23, 24, 25, 26
            31, 32, 33, 34, 35, 36
            41, 42, 43, 44, 45, 46
            51, 52, 53, 54, 55, 56
a[2][3] = 34
*(a[2]+3) = 34
*(*(a+2)+3) = 34
*(*a+2*n+3) = 34
```

It is true no two words are exactly the same. Synonyms are no exception. Here in Programme 6 both $a[i][j]$ and $*(*(a+i)+j)$ mean the same thing, the latter is more efficient than the former. This is because multiplication is computationally expensive, and $a[i][j]$ where the dimension of a is $m \times n$ actually means $*(\&a[0][0]+n \times i+j)$ whereas there is no multiplication in the case of $*(*(a+i)+j)$.

Bibliography

Steven Holzner. *C Programming.* Brady, 1991